Setting up Flexible and Light Weight Trading Contracts with Enhanced User Privacy Using Smart Contracts

Sina Rafati Niya, Florian Schüpfer, Thomas Bocek, Burkhard Stiller

Communication Systems Group CSG, Department of Informatics IfI, University of Zürich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
Emails: [rafati|bocek|stiller@ifi.uzh.ch], flo.schuepfer@bluewin.ch

***Abstract*—Smart Contracts (SC) extend the applicability of Blockchains (BC) in various decentralized use cases. This work demonstrates the design and implementation of a trading application which, employs SC and Ethereum BC. This Decentralized Application (Dapp) provides flexibility in requesting user Identity (ID) directly by seller/hirer and buyers/renter. To provide trust, deposits are paid by two sides while setting up contracts. WiFi-Direct is the chosen Device to Device (D2D) communication protocol which provides high data rates and secure data transmission. Light-Weight SC are introduced in this work which, use D2D communications for sending sold or rented object's or each party's images, and ID data directly to other party instead of storing them in the public BC to reduce the costs. Evaluations in terms of D2D deployment, transaction costs, and privacy, indicate that this system is time-efficient and manages the process in a cost-efficient fashion without the need to store and publish all of the user's ID information in BC.**

***Index Terms*—Blockchains, Ethereum, Light-Weight Smart Contracts, Peer-to-peer Applications, Trading Contracts**

I. Introduction

Setting up contracts between two parties require paper work and time dedication along with Identity (ID) declarations. To handle the requested tasks for this purpose, some solutions have been proposed but most of them follow a centralized approach and inherit the drawbacks of centralized architectures by relying on the owners of those platforms as known as Trusted Third Party (TTP). A TTP also can be a single point of failure in each platform. These platforms have to store a description of the purchased items and their price, and customers can interact with that platform to buy an item. Users need to register at each platform and have to give access to their data to the TTP. They also often have to pay transaction fees [1].

By great progress in Blockchains (BC) and relatively the applications and decentralized use cases based on them, there is the possibility to use BC in solving the problem of centralized applications. Specially by emerging Smart Contracts (SC), Decentralized applications (Dapp) are used to automate the already human based applications [2]. To circumvent all the required bureaucracy for setting up trading contracts and to tackle the problems in TTP-based apps, this paper propose a Dapp, based on Ethereum Blockchain (BC) which enabled by SC [3]. This Dapp is developed for Android-based mobile phones with a P2P architecture [4].

978-1-5386-3416-5/18/$31.00 © 2018 IEEE

Some of the most important factors in designing such a Dapp to automate processes of data and money transactions are "reliable data storage", "issuing decentralized trust", "time-efficient and secure transactions", and "user privacy". In order to overcome the high costs of storing transactions in Ethereum BC, this paper introduces "light weight" contracts. Additionally, data types and units are selected deliberately to result in the least transaction costs. Also, as users can decide themselves about the ID data to be provided for each contract, total costs are decreased by minimizing the data to be stored. Estimation equations are developed to calculate SC costs in advance with a high accuracy.

II. Design and Implementation Overview

A P2P architecture is designed for the proposed Dapp, as shown in Figure (1). Contracts are signed on the users local devices and protected by storing only on internal storage and being encrypted while sending. This Dapp can be utilized with two approaches. In the first approach, users install Ethereum Light Client (ELC) [5] on their Android phones and connect the Ethereum BC directly (Figure 1 - nodes in right side). In the second approach, users connect to an Ethereum full client installed on a workstation *e.g.*, a Laptop (Figure 1 - nodes in left side).

To provide ID information two choices are made possible. The first scheme enables using QR readers by reading while every user owns a QR image as shown in Figure 2-c. The second scheme is to send and receive identity data via WiFi-Direct. The latter method provides flexibility in setting the required ID (*e.g.*, asking for buyer's image in addition to name and email). The second method provides higher privacy as the least possible ID information of users are sent out in BC. Last but not least, users by sending their ID information via WiFi-Direct can store images and documents about each other without saving them in SC to reduce the costs. Figure 2 - a illustrates the information related to one contract which a QR

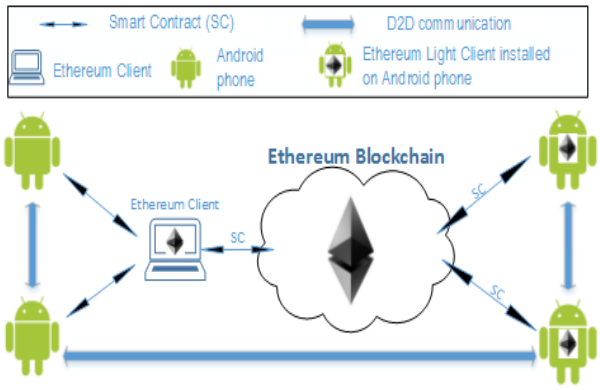

Fig. 1. System Architecture

code is used to read user ID and image of the sold object is also included in contract details.

In this Dapp, service interfaces of the Java library are used by Android Clients (AC) to load and deploy SC. The Java Web3j library is used for wrapping the interface of a SC in a Java class which, is in next step needed by the AC to execute transactions on the BC. Web3j uses the JSON-RPC interface of an external Ethereum client to deploy contracts and interact with them as shown in Figure 2. SC codes in this work are implemented with the Solidity language.

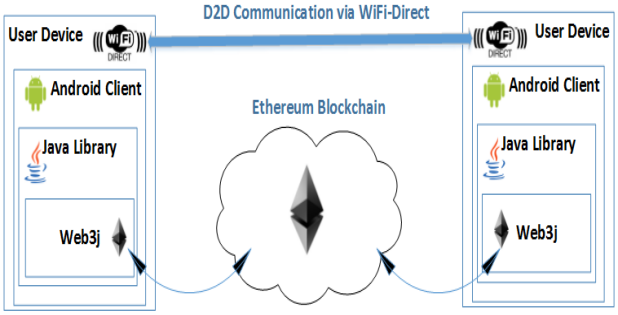

Fig. 2. Implementation Architecture: Integrating Android and Ethereum Clients

Two scenarios of setting up purchase and rental light-weight contracts will be demonstrated in which, users can set up trading SC by Android phones by creating new contracts or, using already existing ones as shown in Figure 2-b. In the rental contract, details of rented object and its price and deposit are included in the constructor of that contract and defined by the hirer. Renters need to pay the deposit in advance by accepting that the deposit only will be accessible after returning back the rented item and paying the rent. In the proposed Dapp, renting price is calculated in "*wei* per second" which is updated regularly based on [6]. Description of each item includes "Title", "Textual Description," and "Image Signatures". Image signatures are determined by using the SHA256 hash function of an image that can be taken by users (e.g., buyer can take a picture of seller while buying a furniture and vice versa).

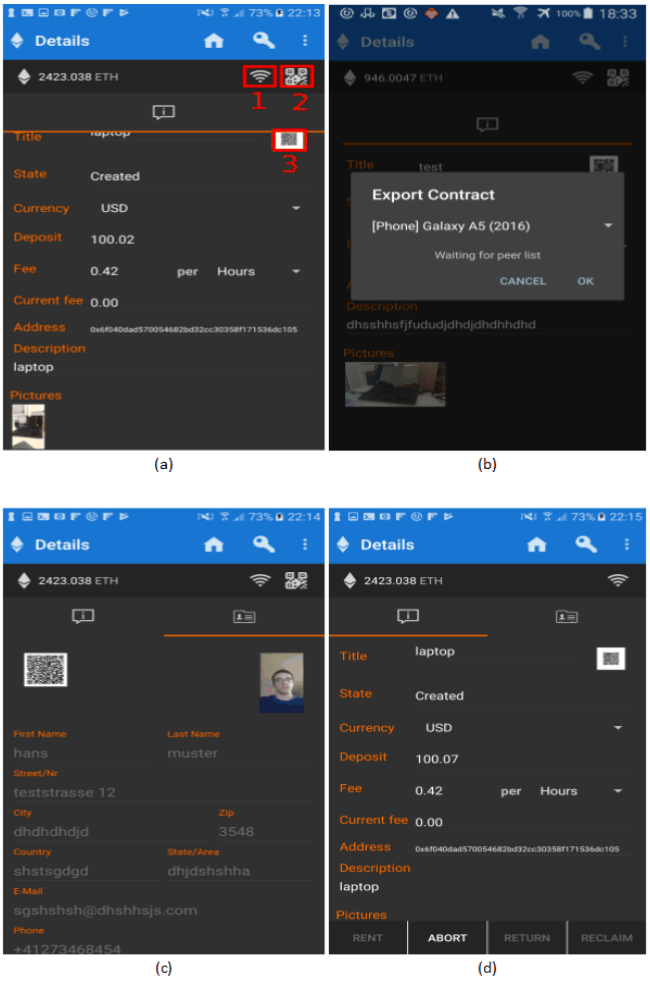

Fig. 3. Process of Purchasing a Contract, (a) Information of a Contract Before Setting Up, (b) Setting Up a Contract, (c) Buyer Identity Information, and (d) Created Contract's Information

REFERENCES

[1] A. Bogner, M. Chanson, and A. Meeuw, "A decentralised sharing app running a smart contract on the ethereum blockchain," in *Proceedings of the 6th International Conference on the Internet of Things*, ser. IoT'16. New York, NY, USA: ACM, 2016, pp. 177–178. [Online]. Available: http://doi.acm.org/10.1145/2991561.2998465

[2] T. Bocek and B. Stiller, *Smart Contracts - Blockchains in the Wings*. Tiergartenstr. 17, 69121 Heidelberg, Germany: Springer, jan 2017, pp. 1–16.

[3] V. Buterin and others:, "What is ethereum?" http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html, Last Seen: 14 Jan, 2018.

[4] S. R. niya, F. Schüpfer, T. Bocek, and B. Stiller, "A peer-to-peer purchase and rental smartcontract-based application," IFI-TecReport No. 2017.04, Zürich, Switzerland, Tech. Rep., nov 2017. [Online]. Available: https://files.ifi.uzh.ch/CSG/staff/Rafati/Purchase-Rental-APP-SC.pdf

[5] "Light Client Protocol," https://github.com/ethereum/wiki/wiki/Light-client-protocol, Last Seen: Dec 25, 2017.

[6] Ethereum.io, "Ethereum avarage gas price chart," https://etherscan.io/chart/gasprice, Last Seen: 14 Jan, 2018.